Convolutional Neural Networks for Driverless Car Simulation

Charlie Davies
The University of Nottingham, School of Physics and Astronomy
(Dated: May 19, 2023)

As a general simplification of the concept, convolutional neural networks (CNNs) are a deep learning technology which processes large datasets, specifically image-based, with ease. By transforming images into normalised numerical matrices, the network convolves over and downsamples the images before applying activations at each layer. Following these steps desired repetitions, the result is flattened into a 1-dimensional vector on which a typical deep neural network architecture is applied. The result of the combination of these technologies produces predictions, as in classical machine learning, for both classification and regression tasks for any number of classes in the given data. In this paper, two challenges employ the use of CNNs for a Kaggle competition and a SunFounder PiCar live demonstration. It will break down the applications of CNNs as well as each aspect of their technology and how they work on a more fundamental level. The concept of transfer learning for CNNs is also explored. The first challenge in the project revolves around training the network for the lowest possible loss and the latter takes into account the performance within a more constricted environment. The approaches to the challenges and the process of developing the final architectures will be discussed and reflected on. The final results of the models demonstrate successful usage of the associated technologies but with some notable difficulties and adjustments throughout. The improvement and future steps for further success and better implementations of deep learning techniques are reflected as an idea for a project based on the results found in this paper.

I. INTRODUCTION

The concept of intelligence itself is still not well-defined and fully understood, however as a general overview it has been described as the computational part of the ability to achieve goals in the world [1]. This is classically applied to biological organisms but in recent decades has become more relevant for computational systems and software which aim to emulate the autonomy of decision-making to reach a certain number of goals. This is called artificial intelligence (AI). Machine learning is a subfield of AI which uses algorithms to make informed decisions based on what it learns from patterns in data [2]. In machine learning, the training of the model is not an explicitly programmed process but is one that has the potential to yield excellent results in distinguishing differences between concepts called classes. Machine learning can be divided into two prominent areas: supervised and unsupervised learning [3]. Supervised learning models train on datasets in which each concept is labelled, e.g. one line of data may correspond to an image of a cat and another to a dog. Where the structure of the data is already known, classification and regression algorithms are applied to make predictions on new data where the label for the data is not yet known. Conversely, unsupervised learning models train on data in which there are no labels. They search for the patterns in the data in an attempt to cluster key differences or select the most relevant variables.

These areas within machine learning can be further divided into many algorithms with differing strengths and purposes. However, this paper solely focuses on the use of convolutional neural networks for image classification, explained later, in the context of driverless car technology. A subfield of machine learning is deep learning [4]. In this category, neural networks are applied to supervised learning tasks by mimicking the human brain using the concept of neurons to learn. Data is received as a 1-dimensional vector and is passed through a fully connected network of neurons, in each of which the sum of the previous connections is taken as the input to a function which determines the value that moves to the next layer. Any number of layers with any number of neurons can be applied to the problem but the final layer will always result in the prediction of a label in the data. An example of a fully-connected neural network is shown in FIG. 1.

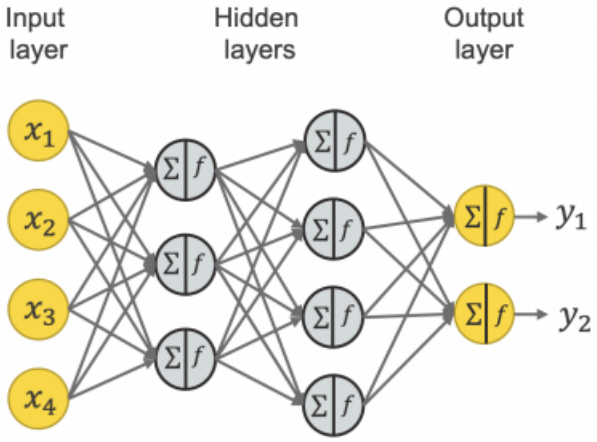

FIG. 1. Fully-Connected Neural Network [5]

Further developing this technology, convolutional neural networks (CNNs) introduce the idea of convolutions

for iterating over image data in preparation for usage in a fully-connected neural network [6].

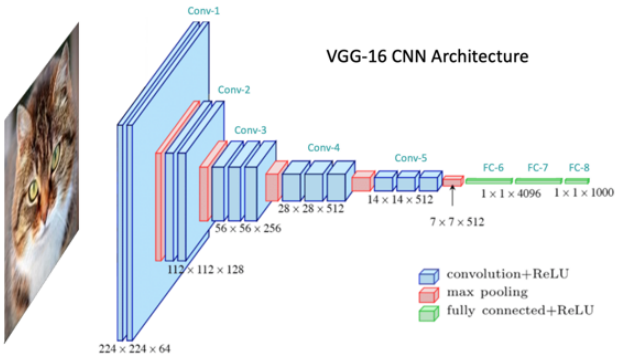

FIG. 2. Convolutional Neural Network [7]

As demonstrated in FIG. 2, the CNN model is divided into convolutional layers, which make use of concepts such as convolutions, pooling and activation functions. Convolutions are integrals which describe the overlap between one function iterated over another [8]. The formula for convolutions is:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

Where f is a kernel, or filter, an input signal g is applied to it. Variable t is a kind of shift and τ is an intermediate variable. In the context of CNNs, the kernel applied is generally a $3 * 3$ matrix applied at various strides over a normalised $n * n$ input matrix representing the initial image. Applying kernels results in a downsampled feature map which is then generally passed through a form of pooling. Pooling is another matrix which takes the maximum of average values in a given portion of the convolution. After this stage, the resulting matrix is typically passed through a ReLU, or Rectified Linear Unit [9], activation function. The ReLU function is mathematically described as:

$$f(x) = max(0, x)$$

For each element in the matrix, the result is the value itself unless it is below zero, for which zero is returned. After these operations, the final result for each convolutional layer is a feature map which represents the features present in the original input in a more condensed, more computationally friendly manner.

After iterating over the desired number of layers, the final resulting matrix is passed into a flattening function which converts a $n * n$ matrix into a 1-dimensional $1 * (n * n)$ vector. At this stage, the result is used as an input for a classical neural network where the final result is the prediction of the labelled classes based on the aforementioned condensed features present in each image. As usual, summations and activation functions are applied, these may not necessarily be ReLU functions, but a new concept is introduced. Regularisation [10] allows the model to deactivate a certain percentage of neurons if the model is found to be overfitting [11]. Overfitting describes the scenario in which the model performs so well on the training data that it handles unrecognised patterns in unseen data poorly.

The final concept this paper covers in CNNs and deep learning is transfer learning [12]. This method takes advantage of models trained on large amounts of data for particular tasks and aims to finetune the already provided weights in the network for a different task. This aims to make training more efficient, improve the accuracy of predictions and aid in tasks where only small amounts of data are available. There are several types of transfer learning: domain adaptation, domain confusion, multitask learning, one-shot learning and zero-shot learning [12]. Each of these approaches uses different amounts of input data depending on the type and number of class(es) the network is to make predictions on. For the sake of brevity, the transfer learning method applied in this paper is discussed in future sections.

All of the concepts mentioned prior have been applied to a project involving two challenges in this paper. The first challenge was developing a CNN for driverless car simulation. The dataset used consists of images of roads, pedestrians and objects where the classes are driving speed and the angle at which to turn given the data in the image. Performance in this challenge was measured on Kaggle against other teams competing for the models producing the lowest loss values, i.e. the smallest difference between the model's predictions and the actual values in a small testing set. The second challenge was to deploy the solution, or a variation of it, onto a SunFounder PiCar-V V2 [13]. The idea of this challenge was to focus more attention on the resource management aspect of model deployment where the performance of the algorithm had to be balanced with the resources available on the Raspberry Pi 4 [14].

II. BACKGROUND

In a wider context, deep learning is applied to many industries and fields ranging from medical imaging [15] and particle physics [16] to cyber security [17] and even agriculture [18]. Perhaps most notably in recent years, CNNs have been at the forefront of driverless car technology [19] [20] [21] [22]. There are a large number of applications for CNNs in driverless car technology. It can be applied to object detection [23], image segmentation [24], pose estimation [25] and image up-scaling [26]. Object detection involves classifying instances of objects and localising them to a bounding box. Similarly, image segmentation is the process of dividing an image into segments. Pose estimation uses the pose and

orientation of an object to track it through a series of images or videos. Finally, image up-scaling, or super-resolution, is the process of enlarging an image without any loss of quality. Further to concepts in CNNs, a large number of prebuilt architectures have been developed by large companies such as Google [27] and Microsoft [28]. These are known as transfer learning models which provide the option of reapplying a fine-tuned architecture and its weights for applying to a new problem. All of these concepts are demonstrably useful for driverless car simulations. As an example, Tesla's full self-driving software utilises eight cameras which identify obstacles, motion, lanes, roads, and traffic lights [29]. They use the data collected from their cars as a continual stream of training data. Whilst the cars employ the computer vision techniques mentioned above, their software is further integrated with other deep learning technologies such as transformers and attention models, all while running at a 50-millisecond response time [29]. This speaks to the power and capability of deep learning and convolutional neural networks. This paper attempts to emulate these ideas at a fraction of the computational and resource scale as a demonstration of what can be done even with a simple model in a relatively short period.

III. METHODOLOGY

After a light preprocessing of the data by removing corrupt images and normalising the image values via the *ImageDataGenerator* class [30] in TensorFlow, the approach to the first challenge was to start with a simple custom CNN architecture. After observing poor performance the next approach was to increase the number of layers and neurons in the network to measure the effect on the results. After several revisions and exploring high layer counts, high neuron counts, utilising dropout and several different activation functions, this approach led to a stagnation of the loss. After struggling to find ways to move forward, the project was moved to Kaggle Notebooks after discovering some issues with the Google Cloud solution. At this point, the model performed better but was not enough as a viable solution for submission. Removing the complexity of the architecture, as shown in FIG. 3, also did not appear to improve the performance.

When researching transfer learning the Inception-ResNetV2 [31] architecture was discovered.

This architecture, as shown in FIG. 4 was imported and attached to the custom solution. This model provided the best result for the Kaggle competition. Before deploying it onto the SunFounder PiCar, new data was collected by running the car through each of the 12 testing scenarios, recording the images along with associated speed and angles. The tests were as follows for the tracks shown in FIG. 5:

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_203 (Conv2D)         (None, 222, 222, 32)      896

 conv2d_204 (Conv2D)         (None, 220, 220, 64)      18496

 max_pooling2d_4 (MaxPooling  (None, 110, 110, 64)     0
 2D)

 conv2d_205 (Conv2D)         (None, 108, 108, 128)     73856

 max_pooling2d_5 (MaxPooling  (None, 54, 54, 128)      0
 2D)

 conv2d_206 (Conv2D)         (None, 52, 52, 256)       295168

 max_pooling2d_6 (MaxPooling  (None, 26, 26, 256)      0
 2D)

 flatten (Flatten)           (None, 173056)            0

 dense_3 (Dense)             (None, 2048)              354420736

 dropout_2 (Dropout)         (None, 2048)              0

 dense_4 (Dense)             (None, 512)               1049088

 dropout_3 (Dropout)         (None, 512)               0

 dense_5 (Dense)             (None, 64)                32832

 dropout_4 (Dropout)         (None, 64)                0

 dense_6 (Dense)             (None, 2)                 130

=================================================================
Total params: 355,891,202
Trainable params: 355,891,202
Non-trainable params: 0
_________________________________________________________________
```

FIG. 3. Custom Complex CNN Architecture

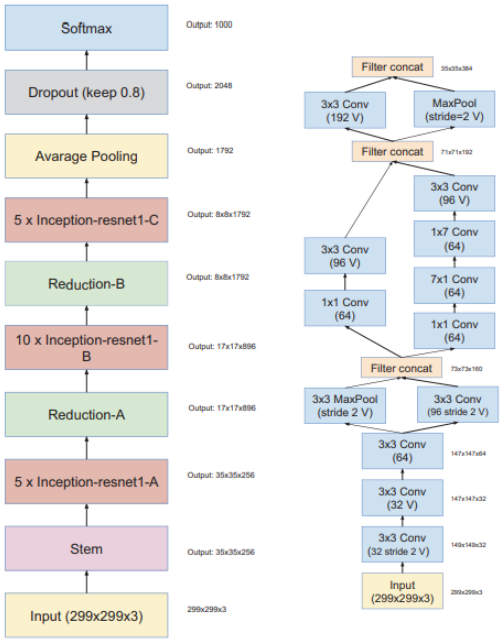

FIG. 4. Inception-ResNetV2 [32]

- Straight drive
- Straight drive, object at side
- Straight drive, pedestrian on track
- Oval drive, both directions
- Oval drive, object at side
- Oval drive, pedestrian on track
- Figure of 8 drive, no stop
- Figure of 8 drive, object in box junction
- Figure of 8 drive, red/green traffic light
- T-junction drive, left turn
- T-junction drive, right turn
- Oval track drive, max speed (50)

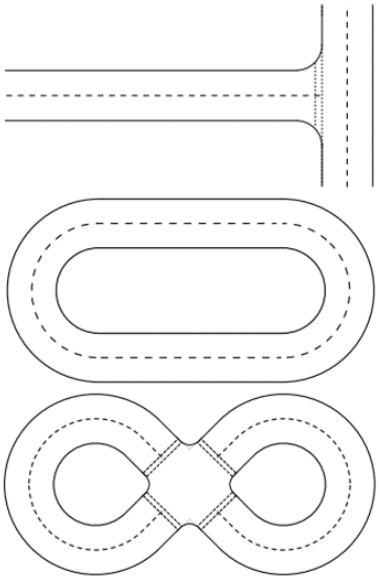

FIG. 5. PiCar Testing Tracks

Attempting to deploy the Inception-ResNetV2 model on the car proved fruitless since it did not move in any of the tests. Further research into transfer learning was conducted and MobileNet [33] was found to be the best-performing model. The architecture for this is shown in FIG. 6.

Thanks to its use of depth-wise separable convolutions [34], MobileNet uses few parameters in the network, filtering through input channels at each layer independently which is less expensive. Aside from inference times of approximately 1200ms, this option performed best and was used as the final implementation.

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

FIG. 6. MobileNet [33]

IV. RESULTS

The quality of performance for the Kaggle competition is measured by both public and private scores based on the loss value of the model. The loss value is determined by the predictions of the model, for speed and steering angle, against the true values in small a testing set of images provided. The private score is based on the performance of the public score and determines the final success of the model for another testing set withheld during the training process. The best performance was generated using the Inception ResNet-V2 transfer learning model. After revising the architecture, the model successfully improved the public score by 0.0156 with a difference of 49% from the original 0.0316 for this model. The difference between the public and private scores is 0.0014, suggesting that overfitting was not an issue in this instance. From an architectural perspective, the second-best performing model was the more complex variation of a custom CNN solution. The final public score was 0.2764, an improvement of 98.7% over the original 2.6007. There is a somewhat minor difference of 0.2044 between public and private scores suggesting a minor issue with the quality of fit. This model started with the worst performance overall but improved the most drastically throughout testing. Finally, the first submission started at 0.2897. The complications with this model are discussed in the following section. Once again, there is a small difference of 0.0126 between the public and private

scores, suggesting little issue with the fit of the model against the data.

Submission	Public	Private
Simple CNN	0.2897	0.3023
Complex CNN	2.6007	2.3963
Complex CNN (Revision 1)	0.2760	0.2839
Complex CNN (Revision 2)	0.2764	0.2848
Inception-ResNetV2	0.0316	0.0273
Inception-ResNetV2 (Revision 1)	0.0160	0.0146

V. DISCUSSION

In many aspects, the project was successful. Achieving 4th place with Inception-ResNetV2 in the Kaggle competition with a model that does not overfit and has a rounded loss of 0.01 is reasonable for the task at hand. There were several setbacks during development which hindered wasted some time but ultimately the impact was minimal. Before understanding and dealing with an issue with Google Colab, and its storage counterpart Google Drive, a focus on complicating the architecture received too much focus and time. After trying to alleviate the poor performance of the Simple CNN with complexity, it was realised that Google Drive stores data in the order of 1, 10, 100, 110, 1000, 1100, etc. Google Colab also performed very poorly in training times without a subscription. In light of these issues, Kaggle Notebooks' proved a more viable solution for storage, programming and training resources. Another issue encountered was during image preprocessing. The original method of storing the data in an array effectively doubled the required storage and loaded far too slowly to be useful. After moving to Kaggle Notebooks and adjusting the method for processing the dataset, the issues with high scores and complexity was overcome so more focus could be put on developing the final revision of the model. In the second challenge, further issues were encountered and A lot was left desired in terms of performance. The first issue encountered was a Bad Marshal Data error produced by Keras. A portion of time was dedicated to understanding this. It eventually became apparent that this is caused by a version mismatch between the Python and Keras. After forcing Kaggle Notebooks to run a more up-to-date Python interpreter, the issue was solved. The second difficulty arose when switching from Keras' .h5 file format to .pb. Some of the files were omitted when downloading the outputs from Kaggle Notebooks. In reflection of the above, some observations were made based on the results and development process. It appears apparent that the best-performing model in a simulated environment may not be the best for deployment in the real-world scenario. Deep learning requires some applied intuition to a lot of trial and error. Keeping the project simple before complicating architectures is key to ensuring meeting deadlines and fixing issues sooner. It would have been interesting to see how much the newly collected data would have changed the result on the SunFounder PiCar since it was not used. Finally, mitigating issues with resource management and minimising inference times in the live demonstration requires decreasing the image size. This is what ultimately lead to the live demonstration failing many of the tests.

VI. CONCLUSION

To summarise, the project was fairly successful in implementing convolution neural networks for driverless car simulation. In the first challenge, the Kaggle competition, the developed model performed well and the final architecture gave a relatively low loss score after training on the data. In the second challenge, the SunFounder PiCar live demonstration, the majority of testing scenarios were passed. Each challenge presented several difficulties, most notably during the second challenge, but each of these was approached with workarounds and fixes to ensure the project moved forward. The final solution for the Kaggle competition left little room for improvement but the difficulties with cloud platforms held the project back at various stages which may have hindered the final score. However, the live performance during the live demonstration could have been improved with some consideration of the newly collected data and image resizing. To improve scores for the final result outside of the project scope, employing image segmentation and object detection could provide the SunFounder PiCar model with a more realistic and flexible approach to driverless car simulation. Further to this, investigating the second MobileNet model [35] could also lead to better performance. Measurements between both with and without image segmentation and object detection could prove interesting.

[1] IBM, (n.d.).
[2] IBM, (n.d.).
[3] . Libesa, Machine learning: A brief breakdown quantdare (2016).
[4] MathWorks, (n.d.).
[5] K. Melcher, A friendly introduction to [deep] neural networks (2021).
[6] IBM, (n.d.).
[7] LearnOpenCV, (2023).
[8] W. MathWorld, (n.d.).
[9] J. Brownlee, A gentle introduction to the rectified linear unit (relu) (2020).
[10] P. Gupta, Regularization in machine learning (2017).
[11] IBM, (n.d.).
[12] R. Bhagyashree, 5 types of deep transfer learning (2018).
[13] SunFounder, (n.d.).
[14] R. Pi, (n.d.).
[15] J. Ker, L. Wang, J. Rao, and T. Lim, Deep learning applications in medical image analysis, IEEE Access **6**, 9375–9389 (2018).
[16] D. Bourilkov, Machine and deep learning applications in particle physics, International Journal of Modern Physics A **34**, 1930019 (2019).
[17] *Deep Learning Applications for Cyber Security* (SPRINGER, 2020).
[18] L. Santos, F. N. Santos, P. M. Oliveira, and P. Shinde, Deep learning applications in agriculture: A short review, Advances in Intelligent Systems and Computing , 139–151 (2019).
[19] N. Sanil, P. A. venkat, V. Rakesh, R. Mallapur, and M. R. Ahmed, Deep learning techniques for obstacle detection and avoidance in driverless cars, 2020 International Conference on Artificial Intelligence and Signal Processing (AISP) 10.1109/aisp48273.2020.9073155 (2020).
[20] Q. Wu, J. Liu, and M. Feng, Msdb-based cnn architecture for image dehazing in driverless cars, 2023 IEEE 3rd International Conference on Power, Electronics and Computer Applications (ICPECA) 10.1109/icpeca56706.2023.10076095 (2023).
[21] N. De Rita, A. Aimar, and T. Delbruck, Cnn-based object detection on low precision hardware: Racing car case study, 2019 IEEE Intelligent Vehicles Symposium (IV) 10.1109/ivs.2019.8814001 (2019).
[22] E. A. Varshini, J. Likitha, N. Aswini, A. H. Nandini, and M. Naziya, Traffic signs recognition using r-cnn, 2022 6th International Conference on Intelligent Computing and Control Systems (ICICCS) 10.1109/iciccs53718.2022.9788295 (2022).
[23] J. Brownlee, A gentle introduction to object recognition with deep learning (2021).
[24] Datagen, (n.d.).
[25] M. S. Walia, A comprehensive guide on human pose estimation (2022).
[26] X. Weber, Deep learning based super resolution with opencv (2020).
[27] Google, (2023).
[28] Z. Lenyk and J. Park, Microsoft vision model: A state-of-the-art pretrained vision model (2021).
[29] A. Shah, How tesla uses and improves its ai for autonomous driving (2023).
[30] TensorFlow, (2023).
[31] Keras, (n.d.).
[32] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, Inception-v4, inception-resnet and the impact of residual connections on learning (2016), arXiv:1602.07261 [cs.CV].
[33] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, Mobilenets: Efficient convolutional neural networks for mobile vision applications (2017), arXiv:1704.04861 [cs.CV].
[34] A. Pandey, Depth-wise convolution and depth-wise separable convolution (2018).
[35] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, Mobilenetv2: Inverted residuals and linear bottlenecks (2019), arXiv:1801.04381 [cs.CV].